4T

УДК 004.6(075.8)

В.Н. Терещенко, Д. Янчик, Д. Пустовойтов, Е. Чернышов
Киевский национальный университет имени Тараса Шевченко, Украина
v_ter@ukr.net, razor.den@gmail.com, dmytro.pustovoytov@gmail.com

Подход к поиску оптимального пути между двумя точками на множестве преград

В статье представлен алгоритм поиска путей на плоскости с учетом преград в виде простых не пересекающихся многоугольников, со сложностью *O(nln(n))* и использованием линейной памяти.

Введение

Постановка проблемы. В современных компьютерных играх (зачастую стратегии) разработчики сталкиваются с интересной проблемой: как бы провести некоторые отряды юнитов через некую местность, теряя минимальное количество тактов процессора на вычисления. Ведь если решить проблему с быстродействием такого алгоритма, можно было бы устраивать баталии невиданных масштабов, в полностью интерактивной разрушаемой среде для препятствий любой формы и размеров.

Анализ современных алгоритмов. Одним из самых эффективных алгоритмов поиска пути по графу является А*-алгоритм и его оптимизации [1], [2]. Большинство предложенных оптимизаций этого алгоритма связаны с поиском решения проблемы быстрой и оптимальной обработки большого количества данных и преобразования их к виду, приемлемому для А*-алгоритма. Такая проблема возникает, если игровое пространство непрерывно. При этом позиции объектов и препятствий сохранены в виде непрерывных значений и должны быть настолько точно представлены, как и разрешение экрана. Примером решения этой проблемы могут быть подходы, использующиеся для мобильных роботов. Одним из таких подходов есть сведение непрерывного пространства к нескольким дискретным вариантам [2]. Способы дискретизации пространства включают:

– нанесение ячеистой сетки на поверхность пространства поиска;

– выделение критических точек, расположенных в основном вблизи вершин препятствий обхода;

– разбиение на выпуклые полигоны пространства, не занимаемого полигональными препятствиями; промежуточными точками могут быть центры полигонов или точки на границах полигонов [3];

– разбиение на квадраты, где каждый квадрат, не являющийся однородным, разделяется на четыре меньших квадрата, центры которых используются для поиска пути;

– пространство между смежными препятствиями рассматривается как цилиндр, форма которого изменяется вдоль его оси. Вычисляется ось, проходящая через пространство между двумя смежными препятствиями (включая стены), и эти оси используются для поиска пути;

– представления препятствия в виде потенциального поля [4], сила которого обратно пропорциональна расстоянию до него. Также существует однородная сила притяжения к цели. Через близкие постоянные интервалы времени вычисляется сумма притягивающих и отталкивающих векторов и объект передвигается в этом направлении. Проблема этого подхода состоит в попадании объекта в локальный минимум; существуют различные способы выезда из таких точек.

Целью работы есть построение общего эффективного алгоритма поиска кратчайшего пути, на множестве преград в виде простых многоугольников, оптимизирующего современные компьютерные игры по количеству юнитов в играх жанра стратегии, а также по сложности и разнообразию препятствий в играх жанра екшен.

В работе предложен новый подход решения рассматриваемой задачи, который позволяет реализовывать игры на качественно новом уровне.

1 Основная часть

В основе представленного алгоритма лежит идея интеграции нескольких эффективных методов решения задач вычислительной геометрии и геометрического моделирования для достижения максимально быстрого поиска кратчайшего пути на плоскости, с использованием минимального объема памяти для хранения структур данных.

Постановка задачи. Пусть задано 2 точки – A (стартовая) и Б (конечная). Область поиска (внешний мир) представлена в виде простого многоугольника (рис. 1).

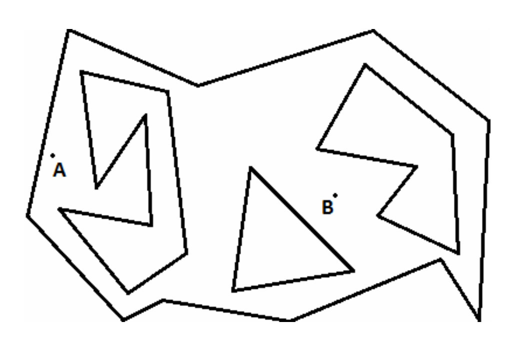


Рисунок 1 – Область поиска – простой многоугольник

Пусть преграды внутри рассматриваемого мира описаны в виде непересекающихся между собой и не пересекающих внешнюю границу мира простых многоугольников (рис. 1). (В случае пересечения следует решить задачу о разделении таких многоугольников на множество простых непересекающихся между собой многоугольников, что не является целью данной работы.) Учитывая решение задачи оптимизации для компьютерных игр, будем задавать вершины парами координат. Пусть многоугольники представляются и хранятся в виде циклически упорядоченной последовательности точек (вершин). Для этого используется $O(n)$ памяти.

Чтобы решить нашу задачу, разобьем ее на пять подзадач: регуляризация графа, выделение монотонных многоугольников, триангуляция монотонных многоугольников, локализация точки, A*-алгоритм. Рассмотрим решения каждой из этих задач.

1.1 Регуляризация графа

Регулярность графа рассматривается относительно некоторой прямой l. Не ограничивая общности, пусть это будет ось OX. Сориентируем граф (рис. 1) по возрастанию в направлении оси OX. Тогда вершину v будем называть *регулярной*, если множества входящих и исходящих из этой вершины ребер одновременно не пустые. Вершина v называется *нерегулярной*, если хотя бы одно из множеств входящих или исходящих из нее ребер пустое.

Граф называется *регулярным* относительно некой прямой (например, оси OY или OX), если все его вершины, кроме первой и последней, регулярны. Если же хотя бы одна из вершин (кроме первой и последней) нерегулярна, граф называется *нере-*

гулярным. *Регуляризация* графа – процесс преобразования нерегулярного графа в регулярный. Граф на рис. 1 не регулярен относительно оси *OX*, и его необходимо регуляризировать. Для этого используем алгоритм плоского заметания [5], в результате чего получим область поиска в виде регулярного графа (рис. 2).

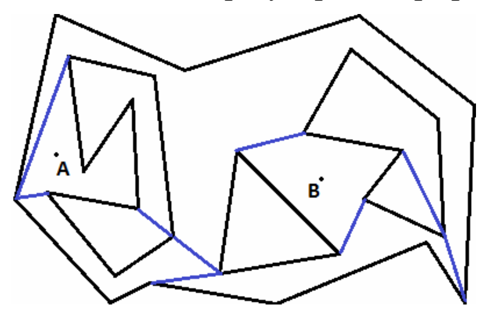


Рисунок 2 – Область поиска после применения алгоритма регуляризации

1.2 Выделение монотонных многоугольников методом цепей

Следующий шаг алгоритма – разбиение полученной области поиска (рис. 2) на монотонные многоугольники. Для этого наиболее подходящим является эффективный алгоритм монотонных цепей, предложенный Ли и Препаратой [6] и усовершенствованный в работе [7], который применяется для решения задачи локализации точки на плоскости. Согласно этому алгоритму, за три линейных прохода по графу (рис. 2) мы получим сбалансированный взвешенный граф, который расцепляется на полное множество монотонных, относительно оси OX, цепей (рис. 3).

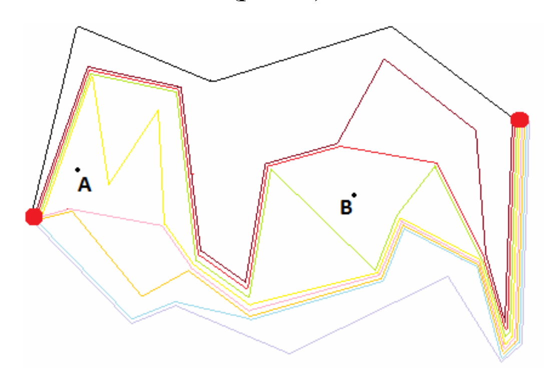


Рисунок 3 – Выделенные цепи

На основе этого множества строится структура данных – взвешенное бинарное дерево монотонных цепей. Эта структура данных позволяет нам за линейное время произвести триангуляцию графа и за *O(logn)* локализировать точку.

1.3 Триангуляция монотонных многоугольников

В этой задаче необходимо триангулировать монотонные многоугольники, образованные множеством последовательных монотонных цепей графа на рис. 3. Полигон называется *монотонным*, если он состоит из двух (верхней и нижней), монотонных относительно одной и той же прямой l, цепей [5].

При выделении многоугольников методом цепей все полученные многоугольники являются монотонными так, как все цепи монотонны относительно оси OX (вертикальное заметание слева направо обеспечивает монотонность). Более детально это описано в [8], [9].

Алгоритм триангуляции монотонных многоугольников

Переименуем вершины нашего монотонного многоугольника в $v_1, v_2, \dots, v_n$ в порядке возрастания координаты абсцисс (именно в таком порядке наш алгоритм будет обрабатывать вершины). Алгоритм хранит стек s вершин, которые были проверены, но не обработаны полностью. Он формирует последовательность монотонных полигонов $p = p_1, p_2, \dots, p_n = 0$. Полигон p_i, как результат обработки вершины v_i, получается путем отсечения нуля или нескольких треугольников от предыдущего полигона. Алгоритм завершает работу с пустым полигоном, а множество треугольников, полученное в процессе обработки, представляет собой триангуляцию исходного полигона p. Пусть $s_1, s_2, \dots, s_n$ – вершины со стека в порядке возрастания снизу вверх, тогда на каждом шаге возможно три варианта их расположения:

1. $s_1, s_2, \dots, s_t$ упорядочены до возрастания координаты x и содержат каждую из вершин полигона p_{i-1}, расположенную справа от s_1 и слева от s_t.
2. $s_1, s_2, \dots, s_t$ являются последовательными вершинами либо в верхней, либо в нижней цепочках полигона p_{i-1}.
3. $s_1, s_2, \dots, s_n$ являются вогнутыми вершинами полигона p_{i-1} (внутренний угол каждой из них более 180^o).

Следовательно, последующая вершина v_i в обработке может быть в трех соотношениях с вершинами $s_1, s_2, \dots, s_t$ (рис. 4):

a) v_i соседняя с s_t, но не с s_1;

b) v_i соседняя с s_1, но не с s_t;

c) v_i соседняя и с s_1, и с s_t.

Ядро полигона – подмножество его точек, из которых виден весь полигон (ядро выпуклого полигона он сам). Полигон называется *веерообразным*, если ядро содержит одну или более вершин – *корней* полигона. Соответственно действия в каждом из трех случаев разные.

В случае *a*: пока $\angle v_i\ s_t\ s_{t-1} < 180^o$, отсекаем полигон $v_i\ s_t\ s_{t-1}$ и уменьшаем t. Когда угол стал больше 180^o либо t стало равным 1, заносим v_i в стек (рис. 4a).

В случае *b:* отсекаем полигон $v_i\ s_t\ s_{t-1} \dots s_1$ – он является веерообразным с узлом в точке v_i, триангулируем его. Заносим в стек s_t, после – v_i (рис. 4b).

В случае *c*: v_i является v_n и полигон $v_i\ s_t\ s_{t-1} \dots s_1$ является веерообразным с углом в точке v_n. Выполняем триангуляцию и заканчиваем алгоритм (рис. 4c).

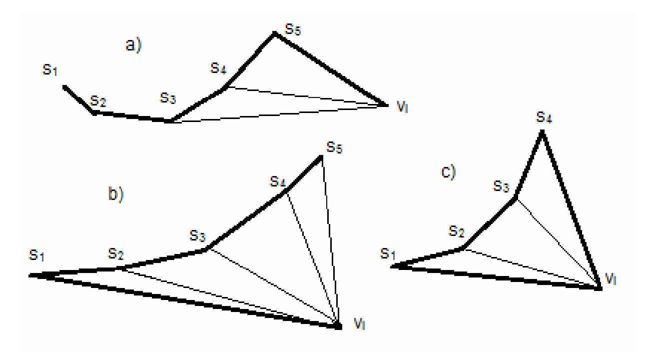


Рисунок 4 – Три случая обработки следующей вершины v_i

В результате получим триангулированный граф (рис. 5). Порядок выполнения шагов триангуляции сверху вниз, слева направо. Более детальное описание алгоритма представлено в работах [4], [5], [9].

1.4 Локализация точек поиска методом цепей

Для локализации точек *A* и *B* будем использовать метод цепей [6] и структуру данных (бинарное дерево цепей), построенную в разделе 2.2. Сделать это достаточно легко, учитывая то, что у нас имеется планарный граф. Запускаем метод цепей для уже триангулированной области поиска. Особенности хранения данных в памяти позволяют бинарным поиском найти, между какими 2 цепями находится наша точка. После этого не составит труда вычислить, в каком треугольнике находится точка и уже можно приступать к применению А*-алгоритма.

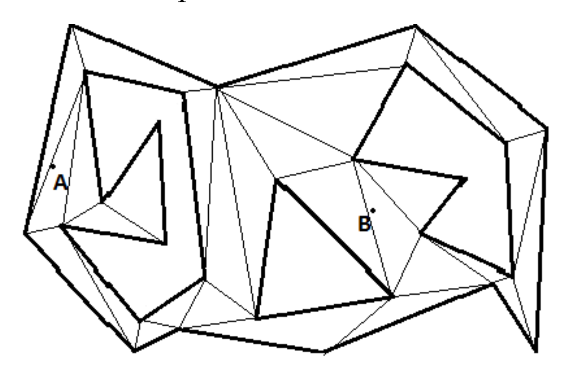


Рисунок 5 – Триангуляция области поиска

1.5 А* алгоритм

А*-алгоритм [1], [2], [10] пошагово просматривает все пути, ведущие от начальной вершины в конечную, пока не найдет минимальный. Как и все алгоритмы поиска, он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. От заданного алгоритма (который тоже является алгоритмом поиска по первому лучшему совпадению) его отличает то, что при выборе вершины он учитывает, помимо прочего, весь пройденный до неё путь (составляющая $g(x)$) – это стоимость пути от начальной вершины, а не от предыдущей, как в жадном алгоритме) [8].

В начале работы рассматриваются узлы, смежные с начальной точкой; выбирается тот из них, который имеет минимальное значение $f(x)$, после чего этот узел раскрывается. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа («множеством частных решений»), которые размещаются в очереди с приоритетом. Приоритет пути определяется по значению $f(x) = g(x) + h(x)$. Алгоритм продолжает свою работу до тех пор, пока значение $f(x)$ целевой вершины не окажется меньшим, чем любое значение в очереди (либо пока всё дерево не будет просмотрено). Из множества решений выбирается решение с наименьшей стоимостью. Чем меньше эвристика $f(x)$, тем больше приоритет (поэтому для реализации очереди можно использовать сортирующие деревья).

Алгоритм А* обходит при этом минимальное количество вершин благодаря тому, что он работает с «оптимистичной» оценкой через вершину. Оптимистичной в том смысле, что, если он пойдёт через эту вершину, то у алгоритма «есть шанс», что реальная стоимость результата будет равна этой оценке, но никак не меньше. Когда А* завершает поиск, то он, согласно определению, нашёл путь, истинная стоимость которого меньше, чем оценка стоимости любого пути через любой открытый узел. Но поскольку эти оценки являются оптимистичными, соответствующие узлы можно без сомнений отбросить. Иначе говоря, А* никогда не упустит возможности минимизировать длину пути, и потому является допустимым.

Предположим теперь, что некий алгоритм *B* выдал в качестве результата путь, длина которого больше оценки стоимости пути через некоторую вершину. На основании эвристической информации, для алгоритма *B* нельзя исключить возможность, что этот путь имел и меньшую реальную длину, чем полученный результат. Соответственно, пока алгоритм *B* просмотрел меньше вершин, чем А*, он не будет допустимым. Итак, А* проходит наименьшее количество вершин графа среди допустимых алгоритмов, использующих такую же точную (или менее точную) эвристику. На рис. 6 показан результат поиска оптимального пути.

Более подробную информацию с разными модификациями и оптимизациями А*-алгоритма можно найти в [10].

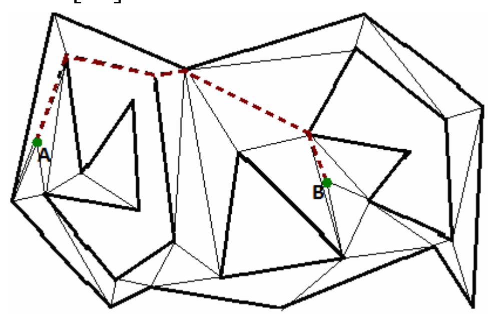


Рисунок 6 – Оптимальный путь с помощью А*-алгоритма в области поиска

2 Анализ сложности алгоритма

Общая сложность алгоритма решения задачи поиска оптимального пути на плоскости с учетом преград в виде простых непересекающихся многоугольников определяется суммой сложностей алгоритмов пяти подзадач.

1. Время, потраченное на регуляризацию n-вершинного планарного графа, составляет $O(nlogn)$ с использованием $O(n)$ памяти [5].

2. Общее время алгоритма выделения монотонных многоугольников в заданном графе равно $O(nlogn)$. При этом $O(nlogn)$ времени уходит на предобработку (сортировка ребер графа за и против часовой стрелки), после инициализация, 1-й и 2-й проходы по $O(n)$ каждый – мы бываем в каждой вершине всего 1 раз. $O(n)$ – сложность выделения многоугольников из последовательности цепей.

3. На триангуляцию монотонных многоугольников уходит $O(n)$ времени. Задаваемый на входе полигон содержит n вершин. Заметим, что при каждой итерации двух внутренних циклов while из стека исключается одна вершина. Однако каждая вершина записывается в стек только однажды (при первой ее обработке) и, следовательно, может быть выбрана из стека тоже только однажды. Поскольку алгоритм выполняет $O(n)$ операций со стеком одинаковой длительности и затрачивает одинаковое время между двумя такими последовательными операциями, то время работы программы пропорционально $O(n)$. Нижняя граница определяется тем, что должна быть обработана каждая из n вершин [11]. Кроме того, в работе [4] показано, что триангуляцию можно оптимизировать до сложности $O(nlog(log\ (n)))$. А Чазелле показал [12], что простой многоугольник может быть триангулирован за линейное время.

4. Локализацию точки в n-вершинном планарном подразбиении можно реализовать методом цепей [6] за время $O(log^2n)$ с использованием $O(n)$ памяти при затратах $O(nlogn)$ времени на предобработку. Учитывая оптимизацию нашего алгоритма под метод цепей, сложность поиска составляет порядка $O(log\ (2\ log\ (n)))$.

5. Временная сложность алгоритма А* зависит от эвристики [13]. В худшем случае число вершин, исследуемых алгоритмом, растёт экспоненциально по сравнению

с длиной оптимального пути, но сложность становится полиномиальной, когда эвристика удовлетворяет следующему условию:

$$|h(x) - h^*(x)| \leq O(\log h^*(x)),$$

где h^* – оптимальная эвристика, то есть точная оценка расстояния из вершин x к цели. Другими словами, ошибка $h(x)$ не должна расти быстрее, чем логарифм от оптимальной эвристики [1].

Выводы

В статье рассматривается новый алгоритм поиска оптимального пути на множестве преград. Представленный алгоритм реализуется с помощью решения пяти подзадач за оптимальное время $O(n\log n)$ (сложность всех 5-х подзадач составляет не более $O(n\log n)$), при этом использует линейную память. Удачный выбор структуры данных (взвешенное дерево цепей) делает алгоритм оптимальным для разработки компьютерных игр различных жанров.

Литература

1. Lester Patrick. “A* Pathfinding for Beginners”. – 21 Oct. 2006 [Электронный ресурс]. – Режим доступа : http://www.policyalmanac.org/games/aStarTutorial.htm.
2. Bryan Stout. Smart Move: Intelligent Path-Finding / Bryan Stout // Game Developer Magazine, October 1996. – P. 28-35.
3. Chiang Y.-J. Optimal shortest path and minimum-link path queries between two convex polygons inside a simple polygonal obstacle / Y.-J. Chiang and R. Tamassia // Internat. J. Comput. Geom. – 1997. – № 7. – P. 85-121.
4. Tarjan R.E. An D(nloglogn)-time algorithm for triangulating a simple polygon / R.E. Tarjan, C.J. Van Wyk // SIAM J. Comput. – 1988. – № 17. – P. 143-178.
5. Препарата Ф. Вычислительная геометрия: Введение / Ф. Препарата, М. Шеймос ; [пер. с англ.] – М. : Мир, 1989.
6. Lee D.T. Location of a point in a planar subdivision and its applications / D.T. Lee, F.P. Preparata // SIAM Journal on computing. – 1977. – № 6(3). – P. 594-606.
7. Edelsbrunner E. Optimal point location in a monotone subdivision / E. Edelsbrunner, L.J. Guibas, J. Stolfi // SIAM J. Computing. – 1986. – № 15(2). – P. 317-340.
8. Скворцов А.В. Применение триангуляции для решения задач вычислительной геометрии / А.В. Скворцов, Ю.Л. Костюк // Геоинформатика: Теория и практика. Вып. 1. – Томск : Изд-во Томск. ун-та, 1998. – С. 127-138.
9. Gilbert P.N. New results on planar triangulations / P.N. Gilbert // Tech. Rep. ACT-15, Coord. Sci. Lab., University of Illinois at Urbana, July 1979.
10. Björnsson Y. Fringe Search: Beating A* at Pathfinding on Game Maps / Y. Björnsson, M. Enzenberger, R.C. Holte, J. Schaeffer // In Proc. IEEE Symp. on Computational Intelligence and Games. – Colchester, Essex, UK, 2005. – P. 125-132.
11. Ласло М. Вычислительная геометрия и компьютерная графика на С++ / Ласло М. ; пер. с англ. В. Львова. – М. : Бином, 1977.
12. Chazelle B. Triangulating a Simple Polygon in Linear Time / B. Chazelle // Discrete Comput. Geom. – 1991. – № 6. – P. 485-524.
13. Pearl Judea. Heuristics: Intelligent Search Strategies for Computer Problem Solving / Judea Pearl. – Addison-Wesley : Addison-Wesley Longman Publishing Co., 1984.

В.М. Терещенко, Д. Янчик, Д. Пустовойтов, Е. Чернишов

Підхід до пошуку оптимального шляху між двома точками на множині перешкод

У роботі представлений алгоритм пошуку шляхів на площині з урахуванням перешкод у вигляді простих багатокутників, які не перетинаються, із складністю $O(n\ln(n))$ і використанням лінійної пам’яті.

V.N. Tereshchenko, D. Yanchik, D. Pustovoytov, E. Chernishov

An Approach to Finding the Optimal Path Between Two Points on a Set of Obstacles

In this paper the algorithm is to find ways on the plane taking into account the obstacles in the form of a self-avoiding polygons and they are finally just did not overlap with the complexity $O(n \ln(n))$ and using a linear amount of memory.

Статья поступила в редакцию 31.05.2010.